UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

DTIC FILE COPY

FOR ANNOUNCEMENT ONLY

PAGE

Form Approved
OMB No. 0704-0188

AD-A226 284

1a. REPORT SECURITY: Uncl

b. RESTRICTIVE MARKINGS: - -

2a. SECURITY CLASSIFI: - -

3. DISTRIBUTION / AVAILABILITY OF REPORT
For Announcement Only
Availability: see # 16

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE: - -

4. PERFORMING ORGANIZATION REPORT NUMBER(S): - -

5. MONITORING ORGANIZATION REPORT NUMBER(S)
ARI Announcement 90-06

6a. NAME OF PERFORMING ORGANIZATION: Cognitive Science Laboratory

6b. OFFICE SYMBOL (If applicable): - -

7a. NAME OF MONITORING ORGANIZATION
U.S. Army Research Institute for the Behavioral and Social Sciences

6c. ADDRESS (City, State, and ZIP Code)
Princeton University
221 Nassau Street
Princeton, NJ 08542

7b. ADDRESS (City, State, and ZIP Code)
5001 Eisenhower Avenue
Alexandria, VA 22333-5600

8a. NAME OF FUNDING / SPONSORING ORGANIZATION: see 7a.

8b. OFFICE SYMBOL (If applicable): PERI-BR

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: MDA903-87-K-0652

8c. ADDRESS (City, State, and ZIP Code): see 7b.

10. SOURCE OF FUNDING NUMBERS

| PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT ACCESSION NO. |
|---|---|---|---|
| 6.11.02.B | 74F | n/a | n/a |

11. TITLE (Include Security Classification)
Facilitating Students' Reasoning with Causal Explanations and Visual Representations

12. PERSONAL AUTHOR(S)
Brian Reiser, Michael Ranney, Marsha C. Lovett, and Daniel Y. Kimberg

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15. PAGE COUNT |
|---|---|---|---|
| Article | FROM 87/01 TO 90/06 | 1989, January, 01 | 8 |

16. SUPPLEMENTARY NOTATION Judith Orasanu, contracting officer's representative
Availability: Reiser, B., Ranney, M., Lovett, M.C., and Kimberg, D. Facilitating (see below)

| 17. COSATI CODES | | |
|---|---|---|
| FIELD | GROUP | SUB-GROUP |
| | | |
| | | |

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Cognitive paychology, Delayed outcome, Decision making, Risk assessment,

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This article reports on a study of students learning to program using GIL, the "Graphical Instruction in LISP" intelligent tutoring system. GIL is designed to explore the construction of explanations from problem solving knowledge, and the use of visual representations in problem solving. We first present a brief overview of GIL, then describe the analyses of students learning to solve simple programming problems using GIL. Our initial data address two questions: 1) Are the explanations provided by GIL effective in guiding student reasoning?, and 2) Are students able to learn to program using graphical representations (i.e. what benefits are provided by this representation of programming?)

16. Supplementary Notation (continued)
students' reasoning with causal explanations and visual representation. in D. Bierman, J. Breuker, & J. Sandberg, Eds. Proceedings of the Fourth International Conference on Artificial Intelligence and Education. (1989). Springfield, VA: IOS.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

| 22a. NAME OF RESPONSIBLE INDIVIDUAL | 22b. TELEPHONE (Include Area Code) | 22c. OFFICE SYMBOL |
|---|---|---|
| Judith Orasanu | 202/274-8722 | PERI-BR |

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

In D. Bierman, J. Breuker, & J. Sandberg, Eds. (1989).
*Proceedings of the Fourth International Conference on Artificial Intelligence and Education.* Springfield, VA: IOS.





# Facilitating Students' Reasoning with Causal Explanations and Visual Representations

**Brian J. Reiser, Michael Ranney, Marsha C. Lovett and Daniel Y. Kimberg**
*Princeton University*
*Cognitive Science Laboratory*
*221 Nassau Street*
*Princeton, NJ 08542*

**Abstract**

This paper reports our study of students learning to program using GIL, the *Graphical Instruction in LISP* intelligent tutoring system. GIL is designed to explore the construction of explanations from problem-solving knowledge and the use of visual representations in problem solving. We first present a brief overview of GIL, then describe analyses of students learning to solve simple programming problems using GIL. Our initial data address two questions: (1) Are the explanations provided by GIL effective in guiding students' reasoning? and (2) Are students able to learn to program using graphical representations (i.e., what benefits are provided by this representation of programming)?



## Introduction

We examine students learning to use GIL, an intelligent tutor for programming. GIL constructs explanations from its problem-solving knowledge and provides visual representations to facilitate students' problem solving (Reiser, Kimberg, Lovett, & Ranney, in press). We present evidence to examine whether GIL's explanations are effective in guiding students' reasoning, and to determine the benefits of its graphical representation in problem solving.

## The GIL Tutor

GIL tutors students learning to write simple LISP programs, employing a problem solver, an explainer, a response manager, and a graphical interface. GIL explains its own reasoning, and provides a visual representation to aid problem solving.

### Deriving Explanations from Problem Solving Rules

Our central aim is to build a rule-based tutor that can construct explanations directly from the content of its problem-solving rules. In GIL, the knowledge used to explain a step in a programming solution is the same knowledge used to reason about it. This method contrasts with other rule-based tutoring systems in which English text for explanations is associated with rules and plans, requiring explanations to be individually prepared for each situation (e.g., Anderson, Boyle, & Reiser, 1985; Johnson & Soloway, 1987). GIL contains a problem-solving model that makes causal knowledge about programming operations explicit, and an explanation component that constructs hints and error feedback from its problem-solving knowledge.

The GIL problem solver includes a set of of reasoning rules and plans. Each rule contains a description of the properties of new intermediate data products that a given LISP step creates. The tutor understands how each step transforms the data so that it might guide the student along novel chains of reasoning. For example, it is not sufficient for the rules to encode the knowledge that taking the *first* of a reversed list will return the last element; the problem solver must allow the tutor to communicate the rationale, i.e., reversing the list causes the last element to become the first, enabling the use of *first* to extract it. To represent this knowledge, each rule describes the properties of the step's input and output. Thus, the problem solver not only knows what step to take, but also why the step is useful. Inferences of this sort can be used to explain why a suggested step is strategic or why a student's step is in error.

GIL constructs explanations in response to errors, bad strategies, or requests for a hint. The GIL explainer draws upon the same knowledge base of rules and plans used by the problem solver. If a hint is requested, the explainer selects a rule that best continues the student's problem solving, suggests the step represented in the rule, and uses the properties of the input and output data to explain why the step is effective. Upon a student's error, GIL generally finds the closest matching rule and uses the description of the input and output to explain how the student's step is discrepant. GIL handles legal errors, in which a student's programming step does not correctly manipulate the selected data, and strategic errors, in which the step is a legal LISP operation but is not useful. GIL responds to errors without any bug catalog, dynamically constructing explanations by comparing students' steps to its own reasoning.

**Visual Representations to Aid Reasoning**

A difficulty in many problem-solving domains is that the syntax of a solution does not reflect the reasoning process required to construct it. GIL was designed to provide a representation for students to communicate with the tutor that was more congruent with the reasoning required in the task (cf. Collins & Brown, 1988). In GIL's graphical programming interface, students build a program by connecting objects representing program constructs into a graph, rather than by defining LISP functions in their traditional text form (Reiser, Friedmann, Gevins, Kimberg, Ranney, & Romero, 1988). Students take a step by selecting a LISP function from the menu and specifying its input and output. Hence, all intermediate products are explicit in the program graph. Rather than specifying only a sequence of operations, such as "the *first* of the *rest* of *(a b c d)*," the student indicates the sequence of transformations, e.g., "get *(b c d)* by taking the *rest* of *(a b c d)*; extract *b* by taking the *first* of *(b c d)*." A complete program consists of a graph specifying how a chain of functions transforms the input data to achieve a particular type of output. Figure 1 displays a partial solution for the program *rotater* (which rotates the last element of a list into the first position). The use of intermediate products makes the process of combining steps more explicit and enables tutoring on individual steps in the solution. With this graphical representation, the structure of the solution mirrors the planning required for program construction. Reasoning chains are represented by branches of the graph used to achieve the final goal. Another advantage is that the intermediate reasoning products are made explicit, so students remain aware of data manipulations as they construct the program. This helps students understand how particular algorithms work, and more generally, helps them learn the logic of embedding functions within other functions to construct an algorithm. Finally, the GIL interface enables students to plan in a variety of directions. GIL's problem solver contains both rules for reasoning forward from the given data toward the goal, and rules for reasoning backward from the goal toward the given data.

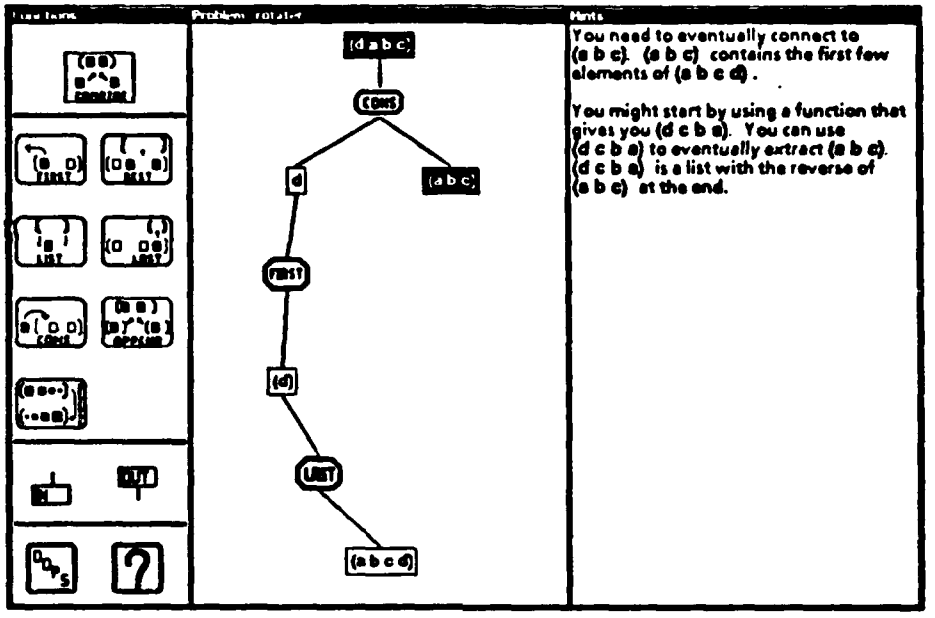


**Figure 1.** A snapshot of GIL's interface en route to a solution for the problem *rotater*, including the first level of a hint.

The interface supports both types of reasoning, and provides a distinct visual representation that mirrors the direction of reasoning. The system supports reasoning about steps in whatever temporal order is natural for the student, even though it may not match the backward, left-to-right order of the final solution's surface form. More details about the *problem* solver, explainer, and interface are presented in Reiser et al. (in press).

## Empirical Investigations of GIL

We investigated the effectiveness of GIL's explanations and visual representations by using GIL to teach simple LISP programming constructs to nine students. All students were Princeton University undergraduates who had never taken a programming course.

### Procedure

The subjects read a short text describing the data structures (atoms and lists), how LISP functions work, how to combine functions into a program, and how to use a chain of functions to describe a general algorithm. The text presented the functions *first*, *rest*, *cons*, *list*, *append*, *last*, and *reverse*. Each function was illustrated with its GIL icon shown operating on some input, producing a corresponding output. The text was adapted from Chapters 1-2 of *Essential LISP* (Anderson, Corbett, & Reiser, 1986). The GIL text substituted graphical representations for the text-based representations and omitted discussions of the syntax for defining functions and referring to variables.

Subjects solved 14 to 15 problems that required generating programs to (a) extract items from lists, (b) manipulate lists, and (c) combine atoms and lists. The solutions varied from three to seven steps. Most problems contained a variety of solutions. For *rotater*, there are six unique solutions, but an average of 78 different paths that a student working with GIL can take to each solution. After each subject read the text's first section, the experimenter demonstrated how to construct a program using GIL. The experimenter showed how to take forward and backward steps, how to select input or output in the graph, how to type in new data, how to cancel partial and complete steps, and how to request hints (like the one shown in Figure 1). All subjects had little difficulty learning to use GIL. No subject asked the experimenter for assistance with the interface beyond the ten-minute demonstration.

**Speed of Solution**

Students read the text and completed the problems in under two hours. This is less than half the time that non-tutored subjects typically spend working on these two chapters. There are many reasons to expect such a difference. Direct comparisons with subjects working without a tutor are difficult to make, however, because the curriculum differs. In standard learning situations, LISP students spend considerable time mastering the syntax of defining functions and using variables. To minimize the load of new material, students are given a full chapter of problems to familiarize them with the data structures and simple LISP functions before introducing function definition. In contrast, GIL students are immediately introduced to the semantics of LISP functions while defining their own functions. Comparisons of overall solution time may be misleading, therefore, because the GIL subjects solve fewer problems (since most of Chapter 1's problems are unnecessary) and are given simplified versions of function definition and variables, which form graphical, rather than textual, structures. Nevertheless, it is informative to compare performance on a common set of five problems from the end of the function definition chapter. These include the *rotater* problem and others of similar complexity. The GIL subjects had been working with LISP at this point for approximately 1.5 hours, and the non-tutored subjects had been working with LISP for about 4 hours. The non-tutored subjects worked in a standard interactive LISP environment, including a simplified screen editor to modify function definitions. These subjects had already defined at least 8 functions, mastering the necessary syntax. The difficulties of the last five problems consisted in planning an algorithm to manipulate the data to produce the desired result. The GIL subjects solved these problems much faster than the non-tutored subjects: an average of 15 minutes versus an average of 58 minutes.

One possibility is that the GIL subjects solved the problems more quickly because they asked for enough hints or received enough error feedback so that GIL essentially told them what steps to take. However, the quantity of explanation does not support this interpretation. Subjects made only .4 errors/problem and requested only .3 hints/problem. Although the error feedback and hints certainly helped the subjects, the four-fold speedup cannot be attributed to GIL solving most of each problem for them. More important factors would seem to be the problem-solving environment and the effectiveness of cogent, timely explanations.

**Direction of Reasoning**

The most striking and important finding in these results concerns the use of forward and backward reasoning. Students were always free to work forward from given data toward any of the goals, or backward from a goal toward the given data. The instructions emphasized this freedom to work in any direction. We analyzed the number of forward and backward steps, excluding "achieve" steps that completed a branch (linking a given with a goal). Subjects showed a strong reliance on forward reasoning, taking 95% of non-achieve steps in the forward direction. Backward steps were used in only 10% of the

problems, and no more than once per problem. Of the backward steps, 92% occurred using LISP combiner operations, thus decomposing the problem into two subgoals. These goal decomposition steps were typically the first step taken by the subject and often followed a hint from GIL to use such a step. When backward decomposition was employed, subjects then worked entirely forward to achieve those goals, eschewing further backward steps.

It is interesting to contrast this directionality with the surface form of final solutions in standard, textual LISP. The code for the solution begun in Figure 1, with the variable *lis* substituted for the example *(a b c d)*, would be *(cons (first (last lis)) (reverse (rest (reverse lis))))*. The order of the functions in the surface form of the code corresponds to a complete backward (top-down) solution, coding *cons* before *first* and *first* before *last*. If subjects' reliance on forward steps in GIL accurately represents their reasoning, then novices do not appear to plan their solution in the order in which the functions appear in the solution. Forcing subjects to enter their code in the outside-in, left-to-right fashion required by standard LISP interpreters forces students into a different strategy than the one in which they can work through an algorithm in the order in which it transforms the data. Thus, a student forced to work within the surface LISP code may be forced to reason in terms of the final solution, rather than the planning necessary to construct the solution. Although expert programmers typically write LISP programs top-down and left-to-right (Anderson, Farrell, & Sauers, 1985), novice reasoners require more search as they fall back on weak methods (Anderson, 1987; Larkin, McDermott, Simon, & Simon, 1980). A tutor that facilitates this search by providing guidance as the student plans is more helpful than a tutor that forces the student to plan entire chains of reasoning before communicating them to the tutor.

Our observations of novice programmers working with human tutors support these findings. Although the tutors encourage students to try portions of their solution on the computer or to enter portions into the editor, students are reluctant to do this. These students often work through the algorithm on paper, or verbally with the tutor, using forward reasoning until a subgoal is satisfied (e.g., getting *(a b c)* in *rotater*). Then, students are willing to enter the newly planned solution, and can do so in the left-to-right order of the required surface form.

The forward reasoning available in GIL enables students to reason in temporal order about a program's data transformations. Given the domain's available operators, general techniques like means-ends analysis would be more effective reasoning from given data to the goal than via backward reasoning, because the search is much more constrained. Since novices generally rely on these weak methods, it helps to provide a representation congruent with this reasoning, so that GIL can provide assistance as the student plans each step. Tutorial guidance is presented precisely on the required step, rather than being delayed until the student has entered the results of the reasoning. The representation also reduces the memory load of keeping track of the properties of the manipulated data.

**Error Feedback and Hints**

Students made more legal errors (in which the chosen function and inputs did not return the indicated output) than strategic errors (legal but useless LISP operations): 83% versus 17%. One interpretation of this finding is that subjects were able to easily generate algorithms for these programs, and that most of the necessary learning involved the operators' semantics, with most errors occurring when operators were misunderstood. However, much of the non-tutored subjects' floundering concerned finding a workable algorithm to satisfy the problem constraints, rather than confusions about the operators. A more likely interpretation, then, is that the GIL environment facilitated algorithm planning, greatly reducing the number of non-strategic steps.

Let us consider whether the causal explanations constructed by GIL were effective. GIL's explainer selects which properties of each object manipulated in a step should be described, and a simple generator constructs sentences to describe each proposition. Do students benefit from these explanations?

Subjects generally understood and accepted GIL's feedback. Subjects were able to immediately fix approximately 50% of the errors described by GIL. Otherwise, subjects either tried a different step, produced another error, or asked for a hint and then took a correct step. GIL's error feedback contains two levels of help. The first points out what is wrong with the student's step (e.g., "remember the input to *append* must be a list") and the optional second level contains specific suggestions about how to fix the step (e.g., "try using *(d)* instead"). Subjects generally found the first level of help sufficient; they asked for more information in only 12% of the errors. Finally, although subjects did not regularly request hints, they generally tried to follow those requested. Subjects attempted to immediately follow 74% of GIL's hints, and were successful in 69% of these attempts.

GIL's environment appears to enhance useful feedback with its explicit intermediate reasoning products. Tutorial feedback can be targeted to the particular parts of a step that is in error. This has been problematic in tutors that treat the surface code (e.g., Johnson & Soloway, 1987; Reiser, Anderson, & Farrell, 1985). For example, if the student uses an inappropriate list-constructing function to combine two products, one of which is the output of the function *last*, the semantic confusion could be in the nature of the list combiner or in the type of product output by *last*. GIL makes all inputs and outputs explicit, so tutorial feedback is focused on the particular properties of the data that are discrepant. This tends to prevent errors from compounding, rendering the problem of diagnosis more tractable.

**Flexibility of GIL**

GIL allows more flexibility than working on a program in top-down, left-to-right order. There are several ways in which GIL lets the student choose the part of the problem on which to work: forward and backward reasoning, selecting any open goal, deleting within a step, and deleting correct steps.

We have discussed the importance of enabling students to reason both forward and backward. Students also take advantage of the opportunity to select which open path to focus on. For example, after a backward *cons* step in *rotater*, subjects might work on achieving the first input, *d*, or the other input, *(a b c)*. GIL does not enforce a left-to-right order — students can choose either goal. Furthermore, once a student starts working toward a goal by taking a forward step, GIL does not force the student to achieve that goal before working on the other goal. Whether this flexibility has a pedagogical benefit is an empirical question that our data do not specifically address. However, our subjects clearly take advantage of this flexibility. Subjects do not always work on the inputs in left-to-right order. There is some tendency to tackle simpler goals before more difficult goals, although subjects sometimes alternate between two goals. Work on one path may lead to the realization that work on another path is needed. This flexibility means that students are not forced to complete a path before additional aspects of the solution are constructed that may constrain choices in the current path.

GIL also allows the student to delete partial or complete steps. Many times, a subject begins a step, then deletes some or all of the partial step. Furthermore, in approximately one problem per subject, one or more completely correct steps were deleted. Although subjects understood that any step completed and accepted by GIL is useful, they occasionally decided to undo a correct step and alter that part of the solution. Without verbal protocols, we cannot be sure of why the subjects changed strategies. However, retrospective interviews indicate that, after starting on one solution path, some subjects realized a "better way" to solve the problem, deleted some of their previous work, and

then pursued their newly discovered strategy. Other subjects mentioned that they became confused about how to continue a current path, but saw another way to solve the problem, and so deleted steps to enable this alternative solution. GIL supports such strategic alterations, which subjects appear to find useful.

The grain size of interaction is coarser in GIL than in the CMU LISP Tutor (Corbett, Anderson, & Patterson; Reiser et al, 1985). In GIL, the student selects a function and specifies its inputs and output before GIL analyzes the step. The CMU LISP Tutor responds to each word typed by the subject; there is no opportunity to reason about the arguments for a function while selecting it, since the tutor responds to the function before the student specifies the arguments. Our GIL subjects' relatively frequent use of the delete key to cancel partial steps indicates that subjects are able to catch many of their minor errors, thus benefiting from the larger units of interaction. Furthermore, the students' ability to change strategies, even at the expense of undoing correct steps, gives them greater control of the interaction. Finally, the specification of the explicit inputs and outputs is particularly important for providing compelling feedback, because the tutor can use them to explain why a step is in error. This use of explicit data avoids situations that sometimes cause students to complain: "It doesn't even know where I want to go with this, why is it already telling me I'm wrong?"

## Conclusions

These preliminary investigations have revealed several promising aspects of our GIL system. Students rely heavily and effectively on the forward reasoning allowed by GIL. The explicit representation of intermediate products helps students reason successfully about their algorithms and construct general solutions using specific examples. The graphical programming representations appear easy to learn and further facilitate students' reasoning. Finally, our results suggest that dynamically constructed causal explanations and visual media offer promising pedagogical advantages — not only for programming, but for many domains in which people design a solution to satisfy various problem constraints.

**Acknowledgments**

We are grateful to Jody Gevins, Patricia Friedmann, Eric Ho, and Antonio Romero for assistance in programming GIL, and to Shari Landes for assistance with data analyses. The research reported here was supported in part by contract MDA903-87-K-0652 from the Army Research Institute and by a research grant from the James S. McDonnell Foundation to Princeton University. The research was performed using equipment donated to Princeton University by the Xerox Corporation University Grant Program. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Army or the McDonnell Foundation.

## References

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review*, 94, 192-210.

Anderson, J. R., Boyle, C. F., & Reiser, B. J. (1985). Intelligent tutoring systems. *Science*, 228, 456-462.

Anderson, J. R., Corbett, A. T., & Reiser, B. J. (1986). *Essential LISP*. Reading, MA: Addison-Wesley.

Anderson, J. R., Farrell, R., & Sauers, R. (1985). Learning to program in LISP. *Cognitive Science, 8*, 87-129.

Collins, A., & Brown, J. S. (1988). The computer as a tool for learning through reflection. In H. Mandl & A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems*. New York: Springer-Verlag.

Corbett, A. T., Anderson, J. R., & Patterson, E. J. (1988). Problem compilation and tutoring flexibility in the LISP Tutor. *Proceedings of ITS-88: The International Conference on Intelligent Tutoring Systems* (pp. 423-429). Montreal.

Johnson, W. L., & Soloway, E. (1987). PROUST: An automatic debugger for Pascal programs. In G. Kearsley (Ed.), *Artificial intelligence and instruction: Applications and methods*. Reading, MA: Addison-Wesley.

Larkin, J. H., McDermott, J., Simon, D., & Simon, H. A. (1980). Expert and novice performance in solving physics problems. *Science, 208*, 1335-1342.

Reiser, B. J., Anderson, J. R., & Farrell, R. G. (1985). Dynamic student modeling in an intelligent tutor for LISP programming. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. Los Angeles, CA.

Reiser, B. J., Friedmann, P., Gevins, J., Kimberg, D., Ranney, M., & Romero, A. (1988). A graphical programming language interface for an intelligent LISP tutor. *Proceedings of CHI'88, Conference on Human Factors in Computing Systems* (pp. 39-44). New York: ACM.

Reiser, B.J., Kimberg, D.Y., Lovett, M.C., & Ranney, M. (in press). Knowledge representation and explanation in GIL, an intelligent tutor for programming. To appear in J. Larkin, R. Chabay, & C. Scheftic (Eds.), *Computer-assisted instruction and intelligent tutoring systems: Establishing communication and collaboration*. Hillsdale, N.J.: Lawrence Erlbaum Associates.